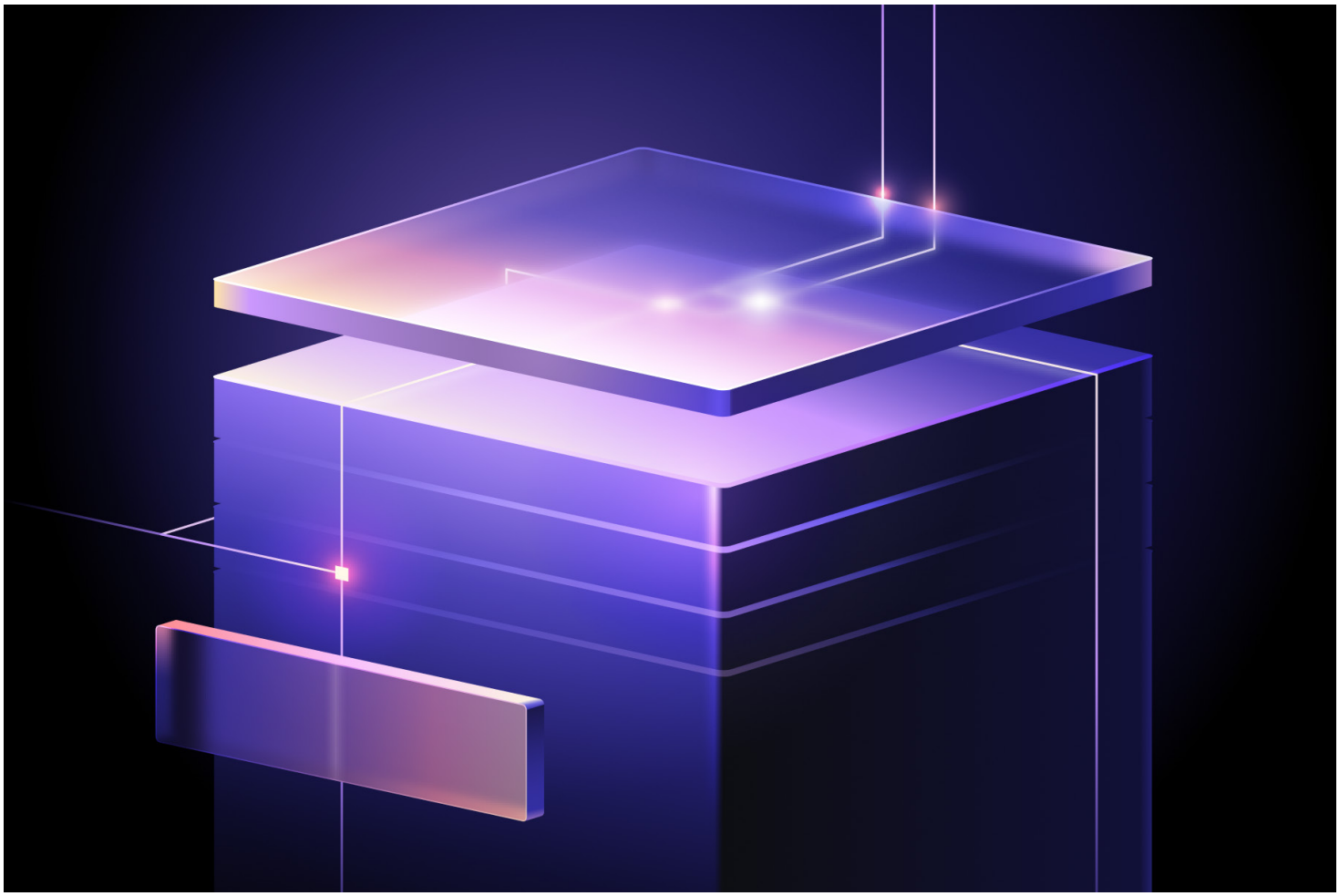


Agentic Lifecycle Management

Redefining cloud operations with agentic workflows and guardrails

HashiCorp
an IBM Company

Contents

1 **Reframing AI as a lifecycle problem**

2 **Day 0 and Day 1: accelerate composition, enforce guardrails**

3 **Day 2+: the context gap and why most AI automation fails**

4 **Make your environment legible: the context layer**

5 **Reference architecture: an agentic fabric for ILM + SLM**

6 **High-value workflows for Agentic Lifecycle Management**

7 **Design principles for governed autonomy**

8 **Maturity model: from copilots to autonomous operations**

9 **A pragmatic plan to start safely**

Executive summary

Generative AI is transforming how teams build and operate cloud infrastructure, rapidly drafting Terraform configurations, proposing remediation plans, and clarifying complex systems. However, organizational success with AI will not be defined by code generation volume. Instead, it will hinge on making accurate, safe infrastructure and security changes throughout the entire lifecycle.

This paper makes a simple argument: AI changes the interface to operations, not the underlying lifecycle. Day 0 and Day 1 work (landing zones and initial deployments) can be accelerated with copilots and code generation. However, Day 2+ operations (patching, drift remediation, incident response, cost optimization, compliance) are where most of an enterprise's effort and risk reside. That is also where AI is least useful unless it is grounded in a trusted context (a unified infrastructure graph spanning clouds, IaC state, and security signals) and governed by policy.

Agentic workflows raise the stakes. When a system executes actions, not just suggests them, governance must be engineered in from the start. Policy as code, identity-based access, and auditable records become the guardrails that make autonomy possible.

Enterprise automation and AI agents mark an identity inflection point: Digital identity extends beyond humans. Service accounts, workloads, automation, and agents operate rapidly, span clouds, and evade legacy IAM visibility. In practice, an agent is a non-human identity (NHI) with credentials and permissions — and the lifecycle is only as safe as the governance of that identity. In an agentic world, operator security encompasses the security of your organization's systems.

HashiCorp's Infrastructure Cloud portfolio is well-positioned for this shift because it already treats lifecycle management as a workflow problem: Infrastructure Lifecycle Management (ILM) to build, deploy, and manage infrastructure over time, and Security Lifecycle Management (SLM) to inspect, protect, and govern identities, secrets, and access from creation through expiry. In an agentic era, these are not separate initiatives — they are a single operating model for safe automation.

1. Reframing AI as a lifecycle problem

Most AI narratives in infrastructure start with technology: models, prompts, copilots, and code generation. That framing is backwards. The right framing is workflow-first: What do teams need to do across the lifecycle, and where can AI safely remove toil without increasing risk?

HashiCorp's lifecycle framing is a useful starting point because it matches how enterprises actually operate:

- **Infrastructure Lifecycle Management (ILM):** Build, deploy, and manage the infrastructure that underpins applications over time — Day 0 (build), Day 1 (deploy), and Day 2+ (manage).
- **Security Lifecycle Management (SLM):** Inspect hybrid environments for unsecured secrets, protect access to systems and sensitive data, and govern identity, secrets, certificates, and privileged access from creation through expiry.

AI and agents are not a replacement for ILM and SLM. They accelerate the same lifecycles by increasing change velocity and expanding how much automation can safely do. The question is whether that acceleration is governed (repeatable, auditable, least-privilege) or uncontrolled (fast drift, fast misconfigurations, and fast breaches).

In other words: If you want autonomous operations, you first need policy-bounded automation controls that continuously enforce compliance, security, and least privilege. The lifecycle platform is the prerequisite.

2. Day 0 and Day 1: accelerate composition, enforce guardrails

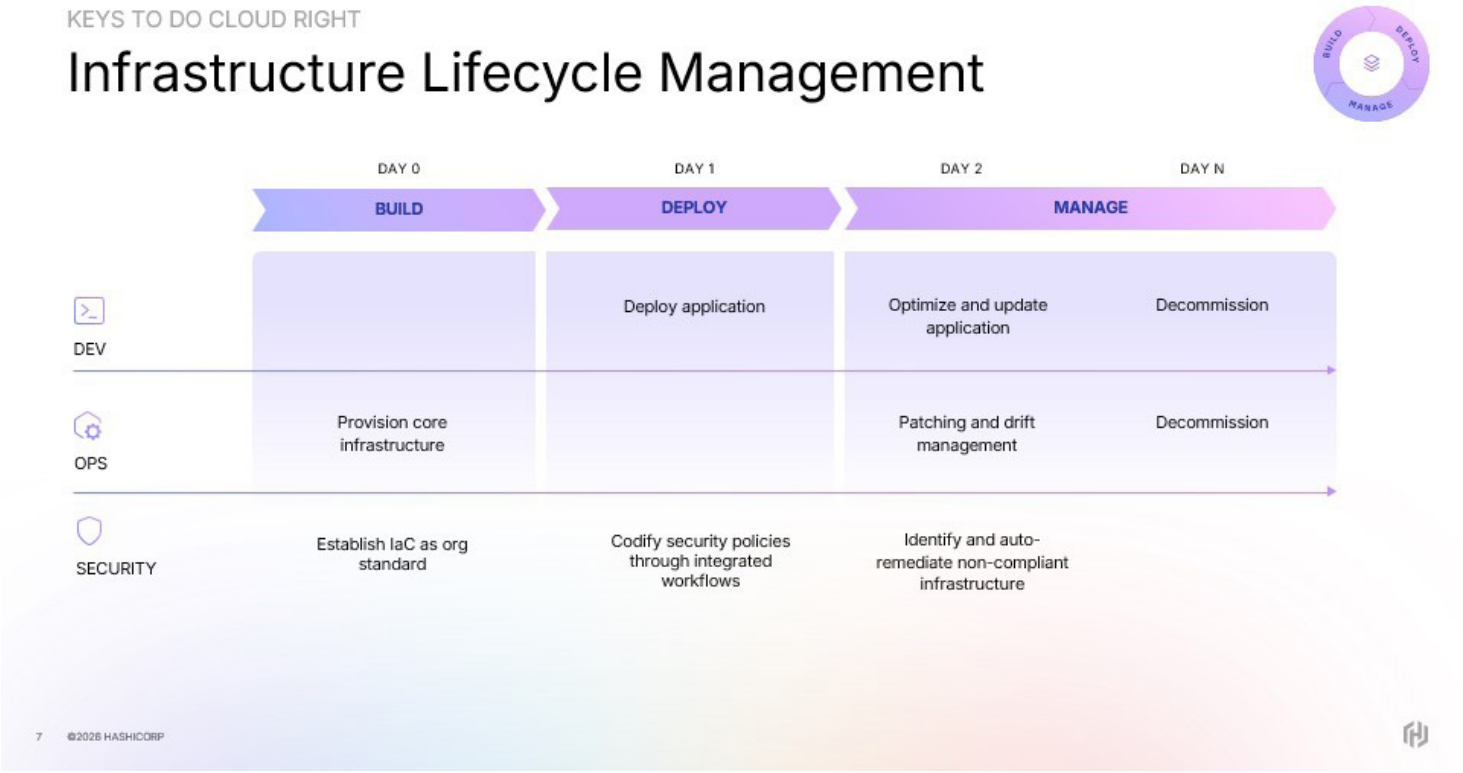
Day 0 and Day 1 are the best-case scenarios for generative AI because they are closest to greenfield.

In a landing-zone build or an initial deployment, the target state is largely defined by public documentation and known patterns. A model can draft a baseline VPC, generate Terraform modules, or suggest golden image pipelines without needing too much context about your specific operating environment.

That said, Day 0/Day 1 AI is only "easy" if you treat its output as draft code. The real risk is that AI enables teams to scale bad defaults faster than security and governance programs can respond.

Practical guidance for Day 0/Day 1 AI enablement:

- Treat AI as an author, not an approver. Humans and automated gates should still validate plans, policies, and security posture before apply.
- Standardize inputs. Require teams to compose from approved modules, providers, and images. If AI is generating from scratch, you are standardizing chaos.
- Make guardrails machine-readable. Use policy as code to enforce baseline requirements (encryption, tagging, network boundaries, approved modules/images) and block unsafe output before provisioning.
- Use AI to scale the governance layer. Policy libraries, documentation, and golden patterns are code. AI can help author and maintain them — but only within your review and testing workflow.



3. Day 2+: the context gap and why most AI automation fails

The moment you move past Day 1, operations become context-dependent. Your environment is no longer a blank slate. There are existing networks, exceptions, legacy applications, active incidents, hidden dependencies, and a history of why the system looks the way it does.

Foundation models do not know that context. Even in Terraform-first organizations, Day 2+ reality includes drift, partial IaC coverage, third-party changes, runtime configuration, and security signals that live outside code. Your environment is private and dynamic. It is not documented publicly or in a public GitHub repository, and it changes every day. Without a way to query the underlying state in real time, AI becomes confident but unsafe as it suggests changes that are plausible, but wrong for your estate.

This is why enterprise value is concentrated in Day 2+. Most operational effort is not in greenfield builds — it is in maintaining, securing, and optimizing what already exists. If agentic automation cannot operate safely in brownfield reality, it will remain merely a demo, not an operating model.

4. Make your environment legible: the context layer

To make agents useful in Day 2+, you need to make your environment legible to AI and machines. That requires a context layer that can reliably and consistently answer questions across tools and environments. This context layer complements, rather than replaces, systems of record like Terraform state, image registries, and security/audit logs by correlating them into a single view of dependencies, permissions, and change history.

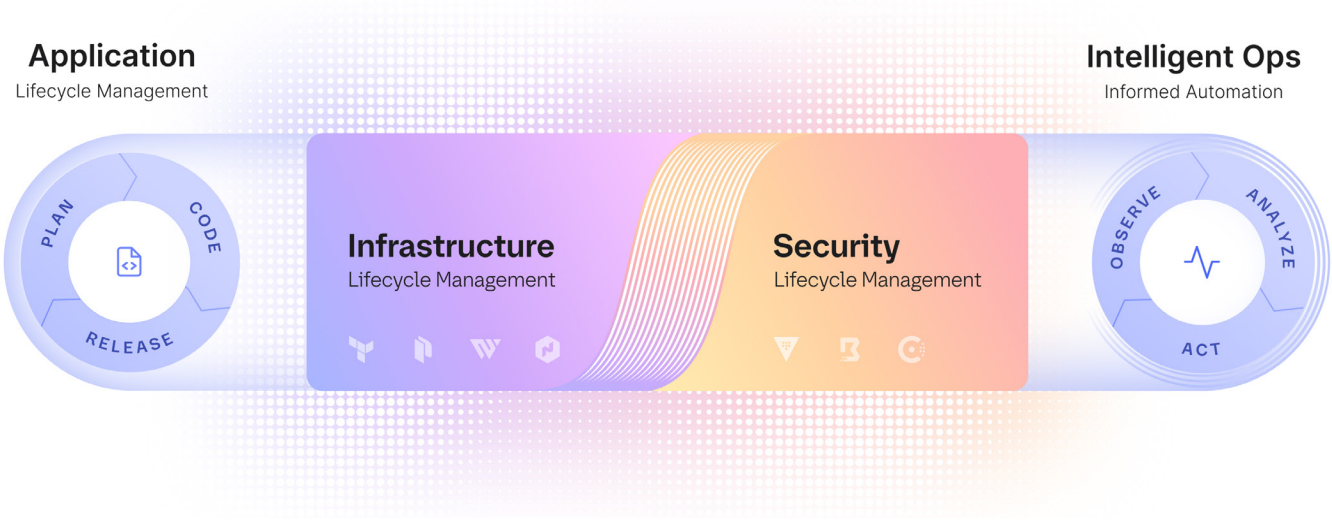
In practice, a context layer does three jobs:

- Ingest and normalize state: Pull inventory and configuration from clouds, on-prem, and tooling (infrastructure state, vulnerability data, identity and audit data, service topology).
- Provide a consistent query interface: A single API or query model that can be used across environments, ideally supporting both structured queries and natural language translation.
- Return evidence, not just answers: For every response, provide the underlying objects, relationships, and sources so a planner (and a human) can validate blast radius and intent.

Opinionated take: Model the context layer as a unified infrastructure graph. A graph is the natural shape for operations because what matters is relationships — which workloads depend on which images, which identities can reach which secrets, and which policies govern which resources. Exposing a single graph query interface (with natural-language translation on top) is how you close the Day 2+ context gap.

In an agentic world, context is not just “what infrastructure exists.” It is also “who (or what) can touch it,” “what secrets and certificates are in play,” and “what has changed recently.” A useful context layer includes identity, policy, and audit signals — not only inventory.

HashiCorp products already provide key systems of record that can anchor this context: HCP Terraform workspaces and state, HCP Packer image registries, Vault audit logs and secrets lifecycle, Boundary session records, and Consul service identity and service-to-service communication. The context layer connects these sources (and third-party tools) into something an agent can reason over safely.

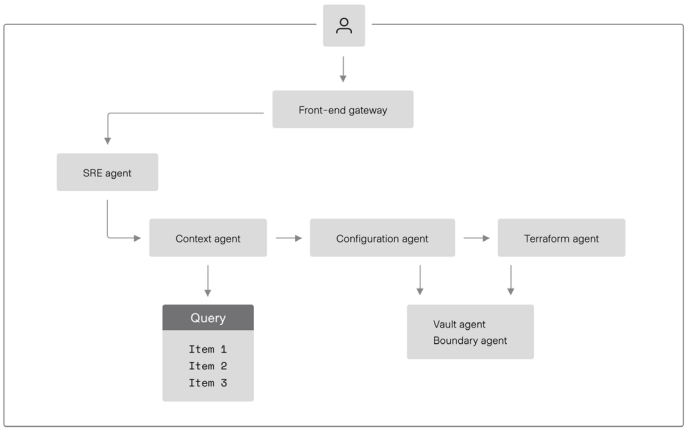


5. Reference architecture: an agentic fabric for ILM + SLM

The enterprise agent pattern that works in practice is not one giant model with broad permissions. It is an agentic fabric: a coordinated set of components that lets software agents plan and execute bounded actions across many tools. In practice, that means a gateway for interaction and guardrails, an orchestrator for planning and state tracking, a context query plane for evidence, and specialist execution skills that run under least-privilege identity and policy controls.

5.1 Core components

- **Conversational gateway:** The user's entry point where intent is captured, identity is verified, and requests are logged. This is where you enforce "who can ask for what" before any tool is invoked.
- **SRE planning agent (orchestrator):** Understands higher-level operational goals, breaks them into tasks, chooses the right specialist agents, and maintains an execution journal (what happened, why, and what evidence supports it).
- **Context agent:** Queries the context layer (ideally a unified infrastructure graph) to understand current state, blast radius, and dependencies. It is the difference between "generate a patch plan" and "generate the patch plan for my actual estate."
- **Terraform agent:** Knows how to inspect and modify IaC safely: identify impacted workspaces, run plans, propose pull requests, and align changes to approved modules and policies.
- **Configuration agent (for example, Ansible):** Generates and runs playbooks to remediate existing infrastructure (Day 2+), using credentials and sessions brokered securely.
- **Vault and Boundary agents:** Provide secrets, credentials, and just-in-time privileged access as part of the workflow, including rotation and revocation. These agents should be least-privilege by default and fully auditable.
- **Policy and compliance gates:** Policy as code enforcement, scanning, and approvals that must pass before actions are executed (and again after execution to validate outcome).



5.2 Why ILM and SLM must be orchestrated together

In the real world, infrastructure rarely exists in isolation. A vulnerability remediation may require patching hosts, updating images, rewriting Terraform modules, rotating secrets, and restarting workloads. These steps cut across ILM and SLM and often require multiple tools. The fabric exists to coordinate these actions safely.

In practice, many workflows are also triggered and validated by observability and security signals. Incidents and degradations are detected through metrics, logs, traces, and posture findings from observability and security tooling. Governed automation needs to pull those signals into the context layer, coordinate ILM + SLM changes, and then close the loop by verifying outcomes in the same telemetry.

This is also where identity becomes foundational: every step in the workflow depends on “can this actor perform this action right now,” with strong logging, just-in-time access, and the ability to revoke access quickly. In agentic workflows, secrets and privileged access are not an implementation detail — they are the safety system.

Agentic automation turns identity into the operational control plane: the policies, credentials, and approvals that govern every action taken by both humans and non-human identities (including agents). Autonomy only scales when identity, secrets, certificates, and privileged access are treated as first-class lifecycles with continuous governance and auditability.

6. High-value workflows for Agentic Lifecycle Management

The highest ROI use cases share three traits: they are repetitive, high-risk if done poorly, and cross multiple tools. These are exactly the scenarios where an orchestrated agentic workflow can reduce toil while improving consistency.

6.1 Vulnerability response (example: OpenSSL)

- Step 1 - Assess impact: Query the context layer to identify affected hosts, images, and dependencies. Produce an evidence-backed impact report.
- Step 2 - Remediate existing systems (Day 2+): Generate a safe change plan and execution playbook (patch, upgrade, configuration updates). Use just-in-time access and short-lived credentials for execution.
- Step 3 - Prevent recurrence (Day 0/Day 1): Update golden images and Terraform modules so future deployments do not reintroduce the vulnerable version. Enforce policies that block noncompliant images/modules.
- Step 4 - Verify and document: Re-query state to confirm remediation, capture audit evidence, and notify stakeholders through existing workflows.

6.2 Suspected pipeline compromise: rotate secrets safely

Secret rotation is a perfect example of an agentic workflow because it is operationally painful, security-critical, and full of dependencies. Rotating credentials is not a single action: It is a lifecycle event that affects applications, infrastructure, and access policies.

- Step 1 - Determine exposure: Identify which credentials could have been accessed by the compromised system (based on policies, audit logs, and dependency graphs).
- Step 2 - Rotate and revoke: Use Vault to reissue short-lived credentials and revoke prior ones. Update policies as needed to reduce the blast radius.
- Step 3 - Propagate change: Restart or redeploy dependent applications so they pick up new credentials; validate that services recover cleanly.
- Step 4 - Prevent recurrence: Improve secret hygiene in pipelines (secret scanning, commit-time detection, and removal of static secrets), and move toward workload identity federation and dynamic credentials.

6.3 Rightsizing and cost optimization with guardrails

Autonomous recommendations are most valuable when the system can prove its work. A mature workflow looks like this: Detect underutilized resources, propose a change with evidence, simulate impact, and execute only within policy and approval boundaries.

- **Detect:** Identify persistent over/underutilization patterns across environments using infrastructure telemetry and cost/utilization insights from FinOps and optimization platforms, in addition to cloud provider metrics.
- **Recommend:** Propose rightsizing or scheduling changes, including expected savings, performance impact, and operational risk.
- **Execute with guardrails:** Apply changes through IaC where possible, using approvals and policy gates.
- **Continuously validate:** Monitor for regressions and roll back automatically if health checks fail.

7. Design principles for governed autonomy

Agentic operations are only as safe as the system that constrains them. The following principles are intentionally opinionated: If you skip them, you may still ship an AI pilot, but you will not be able to scale autonomy responsibly.

- **Context before action:** Agents must query real state and dependencies before proposing or executing changes. If the environment is not legible, autonomy should be limited.
- **Policies are the product:** Guardrails should be expressed as policy as code and enforced automatically. Humans should not be the primary control plane for repetitive governance decisions.
- **Least-privilege by default:** Agents and automation should use just-in-time access, short-lived credentials, and identity-based authorization. Static credentials and broad roles do not scale in an agentic world.
- **Separation of duties:** Authoring, approving, and executing high-risk actions should be separable. Your architecture should support approvals and dual control where needed.
- **Evidence and auditability:** Every action should produce an execution journal: inputs, queries, planned changes, approvals, outputs, and validation results.
- **Bounded autonomy:** Start with narrow, high-confidence workflows. Expand the scope only as policies, context quality, and monitoring maturity improve.
- **Ecosystem-aware orchestration:** Enterprises will never be single-vendor. The fabric should integrate across Terraform, configuration management, ITSM, observability, and security tooling.
- **Humans stay accountable:** Automation can do the work; accountability stays with humans. Design the system so oversight is easy, not a last-minute scramble.

8. Maturity model: from copilots to autonomous operations

Most organizations will progress through predictable stages on their path to autonomy. The goal is not to jump to full autonomy; it is to earn it by improving context quality, standardization, and governance. These stages mirror the adopting, standardizing, and scaling blueprint used across Infrastructure Lifecycle Management (ILM) and Security Lifecycle Management (SLM).

Stage 1 - Adopting: AI as a copilot (drafting)

- Individuals and small teams use AI to draft IaC, policies, and remediation plans that are primarily focused on Day 0 and Day 1 work.
- Execution remains human-driven: Code review, approvals, and applies are still performed by people and existing automation gates.
- Focus on foundations: Establish systems of record (Terraform state, image registries, centralized secrets) and baseline guardrails (policy as code and identity integration).
- Primary risk: accelerating bad patterns faster than governance can respond.

Stage 2 - Standardizing: AI as a governance partner (proposing)

- Platform teams publish approved modules, images, and policies; AI is constrained to compose from golden patterns rather than invent from scratch.
- A context layer is introduced so agents can produce evidence-backed impact analysis and propose changes as pull requests and plans with policy checks.
- Identity and access become standardized for automation: Dynamic secrets, just-in-time privileged access, and auditable sessions are the default path to execution.
- Primary risk: Inconsistent standards or incomplete context leads to brittle recommendations and stalled adoption.

Stage 3 - Scaling: AI as an operator (executing and optimizing)

- An agentic fabric coordinates multi-step workflows across ILM and SLM tooling (Terraform, Vault, Boundary, configuration management, ITSM, and observability) across the hybrid estate.
- Automation expands from human-initiated prompts to system-initiated detection and remediation within defined guardrails (for example: vulnerabilities, drift, secret sprawl, and cost waste).
- Continuous validation, audit evidence, and rollback become standard; humans stay accountable but spend less time on mechanical execution.
- Primary risk: over-trusting automation without strong validation, monitoring, and identity governance.

9. A pragmatic plan to start safely

If you want Agentic Lifecycle Management to work in the real world, start with foundations, not demos. The sequence below is designed to produce one repeatable, governed Day 2 workflow.

Phase 1: Choose one Day 2+ workflow

- Pick a workflow with measurable outcomes (for example: vulnerability response, drift remediation, or secret rotation).
- Define success metrics: time to detect, time to remediate, failure rate, and audit completeness.
- Define guardrails: what the agent is allowed to do, what requires approval, and what is out of scope.

Phase 2: Strengthen systems of record and guardrails

- Consolidate IaC workflows and enforce policy as code gates (approved modules, tagging, encryption, network constraints).
- Ensure secrets and privileged access are brokered through centralized systems (dynamic secrets, session-based access).
- Standardize logging and audit exports so every action can be reconstructed.

Phase 3: Make context legible

- Integrate state sources needed for the chosen workflow (inventory, configuration, IaC state, relevant security data).
- Define a consistent query interface and an evidence model.
- Validate the context layer by running “impact analysis” queries and comparing them to reality.

Phase 4: Implement the agentic fabric

- Deploy a planner + specialist agents pattern with strict permissions and a run journal.
- Wire in policy gates, approvals, and rollbacks.
- Run controlled production pilots and iterate until the workflow is reliable.

In practice, the time required for each phase depends on cloud maturity, change management, and risk controls. Cloud-mature teams may be able to deliver an initial governed workflow in as little as 90 days, whereas more traditional enterprises should expect upwards of 180 days or more for the same sequence, especially when procurement, approvals, and telemetry integration are involved.

The path forward

The question for today's executives is no longer whether AI will become part of cloud operations, but how quickly it can be operationalized without disrupting the business. A unified lifecycle foundation, which integrates Infrastructure Lifecycle Management (ILM) and Security Lifecycle Management (SLM), is essential to sustaining agility, resilience, and trust as workflows become increasingly autonomous.

By adopting Agentic Lifecycle Management, organizations can apply AI to the work that matters most, Day 2+ operations, by pairing autonomy with the guardrails that make it safe through trusted context, policy as code, and identity-based controls for every human and non-human actor. The end state is not just answering human questions but preempting them by shifting from human-initiated prompts to system-initiated optimizations, with humans retaining final authority. Start with one governed Day 2+ workflow, prove it with evidence and auditability, and then scale autonomy as standards mature.

© Copyright IBM Corporation 2026

Produced in the
United States of America
March 2026

IBM, the IBM logo, HashiCorp, Boundary, Consul, Packer, Terraform, and Vault are trademarks or registered trademarks of International Business Machines Corporation, in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on ibm.com/trademark.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

THE INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT.

IBM products are warranted according to the terms and conditions of the agreements under which they are provided.

